# Solving Difficult Game Positions

# Summary

Humans enjoy playing games not only to satisfy their desire for entertainment but also because they seek an intellectual challenge. One obvious challenge in games is defeating an opponent. The AI equivalent to this challenge is the design of strong game-playing programs. Another challenge in games is finding the result of a game position, for instance whether a Chess position is a win or a loss. The AI equivalent to this challenge is the design of algorithms that solve positions. While game-playing programs have become much stronger over the years, solving games still remains a difficult task today and has therefore been receiving attention continuously.

The topic of the thesis is the difficult task of solving game positions. Our research utilizes current developments in the rapidly developing field of Monte-Carlo methods for game-tree search and the continuously evolving field of Proof-Number Search to develop new solving algorithms. To that end, the here described research contributes and tests new *forward-search* algorithms, i.e., algorithms that explore a search space by starting from a game position and developing a search tree from top to bottom. The new algorithms are empirically evaluated on three games, (1) Go, (2) Lines of Action (LOA), and (3) Rolit.

Chapter 1 provides an introduction. It describes the place of games in the domain of AI and provides the notion of *solving games* and *solving game positions*. The chapter introduces the problem statement:

**PS** *How can we improve forward search for solving game positions?*

Moreover, Chapter 1 provides four research questions that address four aspects of the problem statement that are central to solving game positions with forward search. The four questions deal with the following topics, which are part of the recent progress in the research on game-tree search: (1) search with Monte-Carlo evaluation, (2) Monte-Carlo Tree Search, (3) parallelized search, and (4) search for multi-player games.

The aim of Chapter 2 is to provide an overview of search techniques related and relevant to the solving algorithms presented in later chapters. It introduces basic concepts and gives notational conventions. It devotes particular detail to two topics: proof-number algorithms and Monte-Carlo techniques. Proof-number algorithms are stressed because they are well-studied standard techniques for solving. The reason for paying particular attention to Monte-Carlo techniques is that they have recently

contributed substantially to the field of games and AI.

Chapter 3 introduces a new algorithm, MC-PNS, that combines Monte-Carlo evaluation (MCE) with Proof-Number Search (PNS). Thereby the first research question is addressed.

**RQ 1** *How can we use Monte-Carlo evaluation to improve Proof-Number Search for solving game positions?*

An application of the new algorithm and several of its variants to the life-and-death sub-problem of Go is described. It is demonstrated experimentally that given the right setting of parameters MC-PNS outperforms simple PNS. Moreover, MC-PNS is compared with a pattern-based heuristic for initialization leaf nodes in the search tree of PNS. The result is that MC-PNS outperforms the purely pattern-based initialization. We conclude that the reason for the superior performance of MC-PNS is the flexibility of the MCE.

Chapter 4 introduces a novel Monte-Carlo Tree Search (MCTS) variant, called MCTS-Solver, addressing the second research question.

**RQ 2** *How can the Monte-Carlo Tree Search framework contribute to solving game positions?*

The chapter adapts the recently developed MCTS framework for solving game positions. MCTS-Solver differs from traditional MCTS in that it can solve positions by propagating game-theoretic values. As a result it converges faster to the best move in narrow tactical lines. Experiments in the game of Lines of Action (LOA) show that MCTS-Solver with a particular selection strategy solves LOA positions three times faster than MCTS-Solver using the standard selection strategy UCT. When comparing MCTS-Solver to $\alpha\beta$ search, MCTS-Solver requires about the same effort as $\alpha\beta$ to solve positions. Moreover, we observe that $\text{PN}^2$ (a two-level Proof-Number Search algorithm) is still five times faster than MCTS-Solver. Additionally, we show that tree parallelization for MCTS-Solver has a scaling factor of 4 with 8 threads, easily outperforming root parallelization. We conclude that at least during game-play (online) MCTS-Solver is comparable with a standard $\alpha\beta$ search in solving positions. However, for off-line solving positions, PNS is still a better choice.

Chapter 5 introduces a parallel Proof-Number Search algorithm for shared memory, called RP–PNS. Thereby, we answer the third research question.

**RQ 3** *How can Proof-Number Search be parallelized?*

The parallelization is achieved by threads that select moves close to the principal variation based on a probability distribution. Furthermore, we adapted RP–PNS for $\text{PN}^2$, resulting in an algorithm called RP–$\text{PN}^2$. The algorithms are evaluated on LOA positions. For eight threads, the scaling factor found for RP–$\text{PN}^2$ (4.7) is even better than that of RP–PNS (3.5) but mainly achieved because the size of the second-level (i.e., $\text{PN}_2$) tree depends on the number of threads used. Based on these

results the chapter concludes that RP–PNS and RP–PN$^2$ are viable for parallelizing PNS and PN$^2$, respectively.

Chapter 6 focuses on solving multi-player games. The chapter gives an answer to the fourth research question.

**RQ 4** *How can Proof-Number Search be applied to multi-player games?*

The starting point of Chapter 6 is the observation that many two-player games have been solved but virtually no research has been devoted to solving multi-player games. We identify as a reason for this observation that multi-player games generally do not have a unique game-theoretic value or strategy because their search trees have multiple equilibrium points. Therefore, the straightforward way of solving a game by finding the outcome under optimal play (that is applied in two-player games) cannot be applied to multi-player games directly. We therefore propose solving multi-player games under the paranoid condition. This is equivalent to finding the optimal score that a player can achieve independently of the other players' strategies. We then propose Paranoid Proof-Number Search (PPNS) for solving multi-player games under the paranoid condition.

The chapter describes the multi-player variant of the game of Reversi, Rolit, and discusses how to apply PPNS to solve various multi-player variants of $4 \times 4$ and $6 \times 6$ Rolit, and four-player $8 \times 8$ Rolit. The outcome is that in $4 \times 4$ Rolit under the paranoid condition some players can achieve more than the minimum score while in $6 \times 6$ Rolit and in four-player $8 \times 8$ Rolit no player can achieve more than the minimum score. However, we observe that for $6 \times 6$ Rolit and four-player $8 \times 8$ Rolit PPNS performs better than we would expect from standard paranoid search. The chapter concludes by stating that PPNS is able to exploit the non-uniformity of the game tree in multi-player games.

Chapter 7 concludes the thesis and gives an outlook to open questions and directions for future research. While the thesis provides PPNS addressing the class of deterministic multi-player games with *perfect information*, we end by pointing out that solving deterministic multi-player games with *imperfect information* still remains a challenge and we briefly speculate how this problem can be tackled.

# Samenvatting

Mensen genieten van abstracte spelen niet alleen om hun verlangen naar amusement te bevredigen maar ook omdat ze een intellectuele uitdaging zoeken. Een voor de hand liggende uitdaging in spelen is het verslaan van de tegenstander. De corresponderende AI uitdaging is het ontwerpen van sterke spelprogramma's. Een andere uitdaging in spelen is het vinden van het resultaat van een spelpositie, bijvoorbeeld of een schaakstelling een winst of verlies is. De corresponderende AI uitdaging is het ontwerpen van algoritmen die posities oplossen. Terwijl spelprogramma's door de jaren heen veel sterker geworden zijn, blijft het oplossen van spelen nog steeds een moeilijke taak en heeft daarom voortdurend aandacht gekregen.

Het onderwerp van dit proefschrift is de moeilijke taak van het oplossen van spelposities. Ons onderzoek maakt gebruik van actuele ontwikkelingen in het zich snel ontwikkelende veld van Monte-Carlo methoden voor spelboom zoekprocessen en het continu ontwikkelende veld van Proof-Number Search om nieuwe algoritmen te ontwikkelen die oplossen. Het hier beschreven onderzoek draagt bij en test nieuwe *voorwaarts zoekalgoritmen*, dat wil zeggen, algoritmen die een zoekruimte verkennen door vanuit een positie een zoekboom te ontwikkelen. De nieuwe algoritmen worden empirisch geëvalueerd in drie spelen, (1) Go, (2) Lines of Action (LOA), en (3) Rolit.

Hoofdstuk 1 geeft een inleiding. Het beschrijft de plaats van spelen in het AI domein en geeft de begrippen van *het oplossen van spelen* en *het oplossen van spelposities*. Het hoofdstuk introduceert de volgende probleemstelling:

**PS** *Hoe kunnen we beter voorwaarts zoeken bij het oplossen van spelposities?*

Om de probleemstelling te beantwoorden hebben we vier onderzoeksvragen geformuleerd over onderwerpen op het gebied van het oplossen van spelposities door middel van voorwaarts zoeken. Ze gaan over (1) het zoeken met Monte-Carlo evaluaties, (2) Monte-Carlo Tree Search, (3) geparallelliseerde zoekprocessen, en (4) zoekprocessen voor meerdere spelers.

Hoofdstuk 2 geeft een overzicht van zoektechnieken die gerelateerd en relevant zijn voor de zoekalgoritmen die in de latere hoofdstukken worden gegeven. Het hoofdstuk introduceert basisbegrippen en geeft conventies voor notatie. Er wordt dieper in gegaan op twee onderwerpen: Proof-Number algoritmen en Monte-Carlo technieken. Proof-Number algoritmen worden benadrukt omdat ze goed bestudeerde standaard technieken zijn voor oplossen van spelposities. De reden voor de aandacht

voor Monte-Carlo technieken is dat ze een aanzienlijk bijdrage hebben geleverd in het gebied van spelen en AI.

Hoofdstuk 3 introduceert een nieuw algoritme, MC-PNS, dat Monte-Carlo evaluaties (MCE) combineert met Proof-Number Search (PNS). Dit leidt tot de eerste onderzoeksvraag.

**RQ 1** *Hoe kunnen we Monte-Carlo evaluaties gebruiken om Proof-Number Search te verbeteren voor het oplossen van spelposities?*

Een toepassing van het nieuwe algoritme op het leven-en-dood subprobleem in het spel Go wordt beschreven. Er wordt experimenteel aangetoond dat met de juiste parameterinstellingen MC-PNS de standaard PNS overtreft. Tevens wordt MC-PNS vergeleken met een op patronen gebaseerde heuristiek om de bladeren van de PNS zoekboom te initialiseren. Het resultaat is dat MC-PNS de patroongebaseerde initialisatie overtreft. We concluderen dat de reden voor de betere prestaties van MC-PNS is gelegen in de flexibiliteit van de toegepaste MCE.

Hoofdstuk 4 introduceert een nieuw Monte-Carlo Tree Search (MCTS) variant, genaamd MCTS-Solver. Dit heeft ons gebracht tot de tweede onderzoeksvraag.

**RQ 2** *Hoe kan het Monte-Carlo Tree Search raamwerk bijdragen aan het oplossen van spelposities?*

Het hoofdstuk past het MCTS raamwerk aan voor het oplossen van spelposities. De variant MCTS-Solver verschilt van de standaard MCTS aanpak door het feit dat het posities kan oplossen door middel van het propageren van speltheoretische waarden. Als gevolg daarvan convergeert het sneller naar de beste zet in tactische posities. Experimenten in het spel Lines of Action (LOA) laten zien dat MCTS-solver met een specifieke selectie strategie LOA posities drie keer sneller oplost dan MCTS-Solver met behulp van de standaard selectie strategie UCT. Vergelijken we MCTS-Solver met een standaard $\alpha\beta$ zoekproces, dan lost MCTS-Solver LOA posities op met ongeveer dezelfde inspanning als $\alpha\beta$. Bovendien stellen we vast dat $\text{PN}^2$ (een Proof-Nunber Search algoritme bestaande uit twee niveaus) nog steeds vijf keer sneller is dan MCTS-Solver. Daarnaast tonen we aan dat boomparallellisatie voor MCTS-Solver schaalt met een factor 4 voor 8 processorkernen. Wortelparallellisatie wordt hier eenvoudig overtroffen. We concluderen dat gedurende het spel (online) MCTS-Solver vergelijkbaar is met een standaard $\alpha\beta$ zoekproces voor het oplossen van posities. Echter, voor het oplossen van posities offline is PNS nog steeds een betere keuze.

Hoofdstuk 5 introduceert een parallel Proof-Number Search algoritme voor gedeeld geheugen, genaamd RP–PNS. Daarmee beantwoorden we de derde onderzoeksvraag.

**RQ 3** *Hoe kunnen we Proof-Number Search parallelliseren?*

De parallellisatie wordt bereikt door middels een kansverdeling zetten te selecteren in de buurt van de hoofdvariant. Verder hebben we RP–PNS aangepast voor $PN^2$, resulterend in RP–$PN^2$. De algoritmen worden getest op LOA posities. Voor 8 processorkernen is de gevonden schaalfactor voor RP–$PN^2$ (4.7) zelfs beter dan die van RP–PNS (3.5). De reden hiervoor is dat de grootte van de zoekboom op het tweede niveau (dat wil zeggen, $PN_2$) afhangt van het aantal gebruikte processorkernen. Op basis van deze resultaten concluderen we dat RP–PNS en RP–$PN^2$ geschikte technieken zijn om respectievelijk PNS en $PN^2$ te paralleliseren.

Hoofdstuk 6 richt zich op het oplossen van speldomeinen met meerdere spelers. Het hoofdstuk geeft een antwoord op de vierde onderzoeksvraag.

**RQ 4** *Hoe kunnen we Proof-Number Search toepassen voor speldomeinen met meerdere spelers?*

Het uitgangspunt van dit hoofdstuk is de constatering dat veel tweespeler spelen zijn opgelost, maar vrijwel geen onderzoek is besteed aan het oplossen van spelen met meerdere spelers. De laatstgenoemde spelen hebben in het algemeen geen unieke speltheoretische waarde of strategie omdat hun zoekbomen meerdere evenwichtspunten kunnen hebben. Daarom kan de aanpak voor het oplossen van een tweespeler spel niet worden toegepast op een meerspeler spel. Wij stellen voor dit soort spelen op te lossen onder de paranoïde conditie. Dit is gelijk aan het vinden van de optimale score die een speler zelfstandig kan bereiken onafhankelijk van de strategieën van zijn opponenten. We introduceren Paranoid Proof-Number Search (PPNS) voor het oplossen van meerspeler spelen onder de paranoïde conditie.

Dit hoofdstuk beschrijft de meerspeler variant van het spel Reversi, Rolit, en bespreekt hoe PPNS toe te passen om diverse meerspeler varianten van $4 \times 4$ en $6 \times 6$ Rolit, en vierspeler $8 \times 8$ Rolit op te lossen. Het resultaat is dat voor $4 \times 4$ Rolit onder de paranoïde conditie sommige spelers meer dan de minimale score kunnen behalen, terwijl voor $6 \times 6$ Rolit en vierspeler $8 \times 8$ Rolit geen enkele speler meer kan bereiken dan de minimale score. Wij constateren dat voor $6 \times 6$ Rolit en vierspeler $8 \times 8$ Rolit PPNS beter presteert dan het beste geval voor Paranoid Search. Het hoofdstuk sluit af door te stellen dat PPNS in staat is om het non-uniforme karakter van de spelboom in meerspeler spelen uit te buiten.

Hoofdstuk 7 sluit het proefschrift af en geeft een vooruitblik op open vragen en richtingen van toekomstig onderzoek. Hoewel in het proefschrift het PPNS algoritme is voorgesteld om deterministische meerspeler spelen met *perfecte informatie* op te lossen, eindigen we door erop te wijzen dat het oplossen van deterministische meerspeler spelen met *imperfecte informatie* nog steeds een uitdaging is. We speculeren in het kort hoe dit probleem aangepakt kan worden.